Recherche de la validité d'une QBF : exploiter l'insatisfiabilité de la formule propositionnelle sous-jacente

Tony Ribeiro
Université d'Angers
108 route de la Vende
ribeiro@info.univ-angers.fr

Résumé

Nous étudions dans cette article le problème de validité pour les Formules Booléennes Quantifiées (QBF), nous exploitons l'insatisfiabilité de la formule propositionnelle sous-jacente pour restreindre l'espace de recherche lors de la résolution. Par recherche locale stochastique nous tentons d'extraire l'essence des non-solutions : les plus petites combinaisons de variables suffisantes pour falsifier la formule propositionnelle et de les transmettre au fil des générations d'un algorithme génétique. Nous utilisons ensuite les connaissances aquisent par notre algorithme pour guider la recherche de la validité de la QBF.

Mots Clef

QBF, Forme Normale Conjonctive, insatisfiabilité, recherche locale stochastique, algorithme génétique.

Abstract

In this article we study the Quantified Boolean Formula (QBF) validity problem, we use partial insatisfiability of underlying propositional formula to reduce search space size during the resolution. We use stochastic local search to isolate smaller variable combination who falsify the propositional formula and we transmit it by an genetic algorithm. Finally we use this knowledge to guide the validity search of QBF.

Keywords

QBF, Conjonctive Normal Form, insatisfiability, stochastic local search, genetic algorithm.

1 Introduction

Ces dernières années, les prétraitements de formule CNF ont été de plus en plus étudiés par la communauté SAT. Prétraiter une formule CNF avant sa résolution est maintenant considérée comme une étape importante, certaines techniques de prétraitement comme SatElite [5], sont même intégrées aux meilleurs solveurs actuels tel RSAT [8]. Dans ce papier nous nous interessons à la validité des formules booléennes quantifiées (QBF), le problème de validité pour les formules booléennes quantifiées est une généralisation du problème de satisfiabilité pour les formules booléennes. Décider de la validité des QBF est PSPACE-complet, tandis que décider de la satisfiabilité des formules booléennes est NP-complet. C'est pourquoi notre intérêt se porte sur les propriétés de la formule propositionnelle sous-jacente d'une QBF, comme son insatisfiabilité. Dans cet article nous montrons les différentes propriétés de l'insatisfiabilité d'une formule booléenne et l'utilisation qui peut en être faite pour construire une base de connaissance portant sur une QBF. Cette phase d'apprentissage peut être comparée avec le prétraitement des formules CNF cité précédement, à ceci près que notre méthode ne se limite pas à une forme spécifique des QBF, la finalité de nos travaux étant de mettre au point une méthode qui permettrait de guider la recherche de la validité d'une QBF de forme générale. Contrairement à certaines méthodes de prétraitement de formules booléennes nous ne cherchons pas à modifier la structure de la formule, nous ne retirons, ni n'ajoutons de clauses. Notre méthode cherche à isoler les combinaisons de variables suffisants à falsifier la formule propositionnelle, tout en effectuant le moins de calculs possibles sur la formule. La formule n'est réduite que lorsque la valeur d'une variable est fixée par différentes méthodes.

Pour ce faire nous exploitons le potentiel de la recherche locale stochastique, mais contrairement à [6], nous recherchons des non-solutions et tentons d'en extraire ce qui les caractérise pour guider la recherche de solutions. La méthode de prétraitement qui vous est présentée peut également détecter la non-validité d'une QBF, en prouvant l'insatisfiabilité de la formule propositionnelle sous-jacente. C'est à dire en isolant une combinaisons de variables quantifiées universellement, suffisant à falsifier la formule proprositionnelle. Pour nos résultats nous utilisons un solveur basé sur le retour-arrière agrémenté de méthodes exploitant la base de connaissances construite lors du prétraitement de la QBF. La non validité est prouvée lors de la résolution, quand il existe une combinaison d'universelles pour laquelle, il n'existe pas de combinaison d'existentielles satisfiant la formule propositionnelle. La validité

est prouvée s'il existe une combinaison d'existentielles pour chaque combinaison d'universelles, qui satisfait la formule.
En préliminaires nous présentons les quelques définitions nécessaires à la compréhension de nos travaux, puis nous exposons les différentes proriétés de la falsification que nous exploitons. L'algorithme de prétraitement et celui du solveur ainsi que les détails concernant leur utilisation sont expliqués dans la partie suivante. Enfin dans la section résultats est expliqué le cheminement qui nous a conduit à utiliser ces méthodes, cette section contient également des résultats de tests exposant les améliorations concrètes apportées par notre prétraitement.

2 Préliminaires à la résolution de QBF

Afin de définir les différentes méthodes exploitant la falsification de la formule propositionnelle d'une QBF, il est necessaire de rappeler et d'introduire certains termes spécifiques à ces méthodes. Pour ce qui est des symboles employés, $\exists$ représente le quantificateur existentiel et $\forall$ le quantificateur universel, $\neg$ représente la négation, $\wedge$ la conjonction, $\vee$ la disjonction. La constante propositionnelle $\bot$ représente ce qui est toujours faux, $\top$ ce qui est toujours vrai et $[\leftarrow]$ la substitution.

Définition 1 (QBF et Forme Normale Conjonctive) *Une QBF Qn se compose d'une suite de quantificateurs Q et d'une conjonction de disjonctions de littéraux n. Les seuls opérateurs dans une QBF FNC sont $\wedge$, $\vee$ et $\neg$. L'opérateur $\neg$ ne peut être utilisé que dans un littéral, c'est-à-dire qu'il ne peut que précéder une variable. Soit x un symbole propositionnel et σ une QBF.*
La sémantique des QBF est définie par :

$$\exists x\sigma = ([x \leftarrow \bot](\sigma) \vee [x \leftarrow \top](\sigma))$$

et

$$\forall x\sigma = ([x \leftarrow \bot](\sigma) \wedge [x \leftarrow \top](\sigma)).$$

Une QBF est valide si $\sigma \equiv \top$, avec $\equiv$ pour équivalence logique.

Exemple 1 *Cette QBF est sous forme normal conjonctive :*

$$\forall a \exists b \forall c (a \vee b \vee c) \wedge (\neg a \vee \neg b) \wedge (b \vee c)$$

Définition 2 (Scénario) *Un scénario est une fonction à valeur dans BOOL qui donne une valeur booléenne à chacune des variables d'une QBF. Si on représente l'espace de recherche d'une QBF par un arbre dont les noeuds sont les variables et les arcs leur valeurs, un chemin partant de la racine et terminant en une feuille, constitue un scénario.*

Exemple 2 *[1,0,1] est un scénario donnant une valeur pour toutes les variables d'une formule contenant 3 variables, il représente un chemin de l'arbre de recherche.*

Définition 3 (Formules associées à une fonction) *Soit f : $BOOL^n$ -> BOOL une fonction booléenne et x1,...,xn une suite de symboles propositionnels. À la fonction f sont associées respectivement positivement et négativement deux (classes de) formules propositionnelles ϕ_f et ν_f définies sur x1 , . . . , xn qui sont telles que, pour toute valuation v, v satisfait ϕf si et seulement si f (v(x1), . . . , v(xn)) = vrai et v satisfait νf si et seulement si f (v(x1), . . . , v(xn)) = faux.*

Définition 4 (Modèle QBF) *Un modèle QBF étant un ensemble de fonctions booléennes, par la définition 3 qui associe positivement à une fonction booléenne une formule, la formule propositionnelle née de la substitution dans la matrice des symboles existentiellement quantifiés par les formules associées uniquement constituées des symboles universellement quantifiés est une tautologie.*

Un modèle QBF peut également être désigné par les terme « politique », « stratégie » ou « **solution** », c'est ce troisième synonyme que nous utilisons dans cet article.

3 Propriétés de la falsification

L'algorithme qui vous est présenté utilise différentes propriétés de la falsification de la formule propositionnelle sous-jacente d'une QBF afin d'élaguer l'espace de recherche lors de sa résolution. Les méthodes qui vous sont présentées trouvent tout leur sens lorsqu'elles sont combinées.

3.1 Généralisation de scénario

Généraliser un scénario, c'est généraliser une variable de ce scénario. Une variable peut être généralisée si inverser sa valeur dans le scénario ne change pas le fait que le scénario falsifie la formule.

Définition 5 (Valeur générique ?) *Un scénario peut être vu comme une formule propositionnelle, un scénario est alors une conjonction de toute les variables qui ont une valeur spécifiée dans le scénario, la substitution peut donc s'appliquer sur une scénario. Soit une formule F et un scénario S qui falsifie F et v une variable de S. Si [v $\leftarrow \neg$v](S) falsifie F alors v n'a pas d'influence sur le fait que S falsifie F, S est donc généralisable en la variable v et v peut être associée à la valeur générique (notée : ?) dans S.*

Exemple 3 *Soit F une QBF, S, S_1, S_2, S_3 des scénarios tel que :*
$F = \forall a \forall b \exists c (\neg a \vee \neg b) \wedge (a \vee c)$

S = [1,1,1] falsifie $(\neg a \vee \neg b)$ *donc falsifie F*
$S_1 = [c \leftarrow 0](S) = [1,1,0]$ *falsifie* $(\neg a \vee \neg b)$ *donc falsifie F*
$S_2 = [a \leftarrow 0](S) = [0,1,1]$ *satisfait toutes les clauses*

c n'a pas d'influence dans le scénario S, que sa valeur soit 1 ou 0 le scénario falsifie la formule propositionnelle de F donc $S_3 = [1,1, ?]$ *est la généralisation de S en c et* S_3 *falsifie la formule propositionnelle de F.*
S_2 *satisfait la formule propositionnelle, a est necessaire à la falsification de la formule propositionnelle de F par S, on ne peut donc pas généraliser S sur a.*

3.2 Règles d'élagage

Une règle d'élagage représente un scénario falsifiant la formule propositionnelle sous-jacente d'une QBF, une règle est constituée d'une combinaison d'instanciations de variables suffisant à définir une non-solution de cette formule.

Définition 6 (Règle d'élagage) *Une règle d'élagage est une suite d'association (variable = valeur), ordonnée selon l'ordre des quantificateurs d'une QBF dont elle couvre uniquement des non-solutions. Une variable n'apparait au plus qu'une seule fois dans une règle, la valeur associée à une variable ne peut être que 0 ou 1, qui corresponde respectivement aux booléens faux et vrai, le dernier élément d'une règle contient toujours une variable existentiellement quantifiée.*

Définition 7 (Subsume) *Une règle* subsume *une autre règle si tous les éléments de la première règle sont présents dans la seconde.*
Soit R_1 *et* R_2 *deux règles d'élagages, si* $R_1 \subset R_2$ *alors* R_1 subsume R_2*. Si une règle subsume une autre règle, cela signifie que l'espace couvert par la première contient celui couvert par la deuxième, la première règle est une généralisation de la seconde.*

Exemple 4 *Soit F une QBF, S un scénario falsifiant F et* R_1, R_2 *deux règles déduites de S tel que :*
F = (1)
S = [0, 0, 0]
$R_1 = [a = 0, b = 0, c = 0]$ *déduite directement de S.*
$R_2 = [b = 0, c = 0]$ *déduite après* généralisation *de S en a (cf. section 3.1 page 2),* $R_2 \subset R_1$ *donc* R_2 *subsume* R_1*.*

Définition 8 (Couverture) *Un scénario est couvert par une règle si pour chaque variable des éléments de la règle, la valeur dans le scénario correspond à celle de l'élément.*
Soit R une règle et S un scénario, R couvre S si pour chaque élément $[V = val]$ *de R* $[V \leftarrow val](S) = S$*.*
Tout scénario couvert par une règle d'élagage est une non-solution de la QBF sur laquelle porte la règle, par définition des règle d'élagage.

Théorème 1 *Un scénario couvert par une règle d'élagage d'une QBF ne fait pas partie d'une solution de cette QBF.*

Démonstration 1 *Soit une règle d'élagage* R_1 *couvrant uniquement des non-solutions d'une QBF F et soit S un scénario couvert par* R_1*. Si S appartient à une solution de F, alors S ne contient aucune combinaison d'instanciations de variables suffisant à constituer une non-solution de F. Or comme* R_1 *couvre S, tout les éléments de* R_1 *sont représentés dans S et S contient donc une combinaison d'instanciation de variable suffisante pour constituer une non-solution de F, S est donc une non-solution de F comme tout scénario couvert par* R_1*.*

Exemple 5 *Soit S un scénario et* R_1, R_2, R_3 *des règles d'élagage tel que :*
S = [0, ?, 0]
$R_1 = [a = 0, c = 0]$ *couvre S.*
$R_2 = [b = 0, c = 0]$ *ne couvre pas S.*
$R_3 = [a = 1, c = 0]$ *ne couvre pas S.*

3.3 Généralisation de règles

Si deux règles ont la même *arité* (même nombre d'éléments) et qu'elles ne diffèrent que par la valeur d'un élément, alors cet élément n'a pas d'influence dans le fait qu'un scénario soit une non-solution de la formule et les scénarios couverts par ces deux règles sont couverts par leur généralisation.
De même si une règle R_1 dont on change la valeur d'un élément L est subsumée par une règle R_2, alors L n'a pas d'influence dans la falsification de la formule et R_1 est généralisable en l'élément L.

Définition 9 (Généralisation de règle) *Soit une formule F, deux règles d'élagages* R_1 *et* R_2*, L une variable et Lb, Le deux ensembles d'éléments de* R_1*. Si* $R_1 \cup R_2 = [Lb, L = 0, L = 1, Le]$ *alors la valeur de L n'a pas d'impact sur la falsification de F et l'on peut remplacer* R_1 *et* R_2 *par* R_1 *- {L = valeur de L dans* R_1*} (ou* R_2 *- {L = valeur de L dans* R_2*}) dans la base de règle.*

Théorème 2 *La généralisation de deux règles d'élagage couvre tout les scénarios de ces deux règles et ne couvre donc que des non-solutions.*

Démonstration 2 *Soit* R_3 *une règle d'élagage résultant de la généralisation en un élément L de deux règles* R_1 *et* R_2 *ne couvrant que des non-solutions d'une QBF F. Admettons que* R_3 *couvre un scénario S faisant partie d'une solutions de F et dans lequel la valeur de la variable de L est spécifiée (0 ou 1). Or* R_3 *est l'union de deux règles d'élagage sur F, S serait alors couvert par* R_1 *ou* R_2*, et* R_1 *comme* R_2 *ne couvre que des non-solutions de F, donc ne couvre pas S et*

de ce fait R_3 ne couvre pas S.
Soit v la variable de L tel que $[v \leftarrow ?](S) = S'$, et S' un scénario faisant partie d'une solution de F. Par définition de la valeur générique (cf. section 5 page 2) $S' = [v \leftarrow 0](S') \cup [v \leftarrow 0](S')$, si R_3 couvre S' alors $[v \leftarrow 0](S')$ est couvert par R_1 ou R_2, et R_1 comme R_2 ne couvre que des non-solutions, R_3 ne couvre donc pas S'.
R_3 la généralisation de R_1 et R_2 ne couvre que des non-solutions de F et conserve donc les propriétés des règles d'élagage.

Exemple 6 *Soit R_1, R_2, R_3, R_4, R_5 des règles d'élagage tel que :*
$R_1 = [a = 1, c = 0]$
$R_2 = [a = 0, c = 0]$
$R_1 \cup R_2 = [a = 1, a = 0, c = 0]$
$R_3 = [c = 0]$
R_3 est la généralisation de R_1 et R_2 en a, R_3 subsume R_1 et R_2, R_3 couvre donc les scénarios couverts par R_1 et R_2.
$R_4 = [a = 0, c = 1, d = 0]$
$R_3 \cup R_4 = [a = 0, c = 0, c = 1, d = 0]$
$R_5 = [a = 0, d = 0]$
R_5 est la généralisation de R_3 et R_4 en c.

La généralisation de règles permet d'obtenir des règles sans avoir à tester toutes les possibilités d'un scénario et de réduire le nombre de règles. Cette généralisation permet également de créer des règles composées d'un seul élément qui fixe la valeur de la variable concernée par cet élément dans la formule.

3.4 Verrouillage

Si une règle est d'arité 1 alors la seule valeur de son unique élément suffit à définir une non-solution de la QBF, cette valeur n'apparaitra donc jamais dans une des solutions, on peut ainsi simplifier la formule propositionnelle de la QBF en substituant la variable par l'opposé de cette valeur.

Définition 10 (Règle de verrouillage) *Soit v une variable, val une valeur booléenne et $R = [v = val]$ une règle d'élagage d'arité 1, alors aucune solution prouvant la validité de la QBF n'est couverte par R, donc dans toutes ces solutions $[v \leftarrow \neg val]$, R se traduit par la subsitution de v par $\neg val$ dans la formule.*

Lorsqu'une règle de verrouillage est créée on retire de toutes les règles le littéral correspondant à la variable concernée, il n'influence plus la falsification de la formule puisque la valeur de la variable est dorénavant fixée.

3.5 Réduction universelle

Si une règle d'élagage ordonnée selon les quantificateurs se termine par des litteraux portant sur des variables universelles ceux-ci doivent être retirés de la règle. On simplifie ainsi la règle jusqu'à la dernière existentielle de cette règle.

Définition 11 *Soit une règle d'élagage $R = [L1 = val1, ..., Ln = valn, Ln+1 = valn+1]$, Ln porte sur une variable existentielle et Ln+1 porte sur une variable universelle. Si un scénario est couvert par R jusqu'a Ln, alors ce scénario ne fait pas partie d'une solution de la QBF.*

Théorème 3 *Une règle d'élagage dont les derniers élément portent sur des variables quantifiées universellement, est généralisable en ces éléments.*

Démonstration 3 *Soit une règle d'élagage R qui ne couvre que des non-solution et dont le dernier élément L instancie la variable v quantifiée universellement à la valeur val. Soit la règle R_1 constituée de tout les éléments de R hormis le dernier élément L tel que le dernier élément de R_1 L' soit $v = \neg val$. Si R_1 couvre un scénario S_1 faisant partie d'une solution, comme v est quantifiée universellement il y a necessairement un autre scénario S_2 équivalent à S_1 jusqu'à la variable v dans lequel $[v \leftarrow val]$. S_2 est donc couvert par R, or R ne couvre que des non-solutions, S_2 est donc une non-solution et on en déduit que S_1 est également une non-solution. R est généralisable en son dernier élément, par récursivité il en est de même pour tout les éléments en fin de règle portant sur des universelles.*

Exemple 7 *Soit F une QBF, R_1 et R_2 des règles d'élagage tel que :*
$F = \exists a \forall b (a \vee b)$
$R_1 = [a = 0, b = 0]$
$R_2 = [a = 0]$ la réduction universelle de R_1

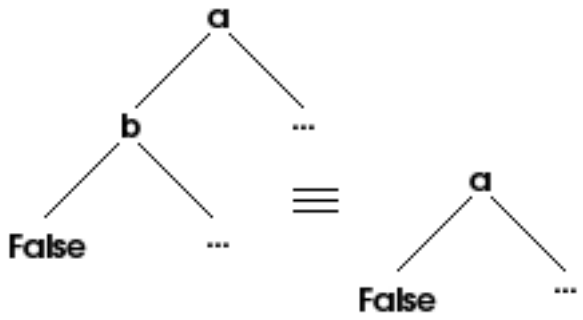

FIGURE 1 – A un noeud correspond une variable, cette variable est substituée par 0 dans le sous-arbre gauche et par 1 dans le sous-arbre droit. Pour qu'un scénario fasse partie d'une solution il faut que les deux branches de l'arbre de recherche partant de b soient à vrai puisque b est quantifié universellement après a, or pour [0,0] on a faux, donc dans aucune solution validant F il n'y aura [a ← 0], R_1 se simplifie en R_2 par réduction universelle.

3.6 Complétude

En combinant la généralisation de règles et la réduction universelle on obtient un ensemble de règles, une base de règle couvrant toutes les non-solutions d'une QBF. La recherche locale stochastique permet de construire tout les scénarios falsifiant directement la formule propositionnelle (générant une clause vide). La généralisation de règles quand à elle permet d'étendre le domaine couvert en se basant sur les relations entre règles d'élagage. Jusqu'ici on ne couvre que des scénarios falsifiant la formule propositionnelle, c'est la qu'intervient la réduction universelle qui nous offre la possibilité de détecter les scénarios satisfiant la formule mais ne faisant pas partie d'une solution de la QBF.
Toutes les non-solutions peuvent ainsi être identifiées et représentées par des combinaisons d'instanciations de variables les plus compactes possibles. Dans l'absolue la base de règles ainsi construite tend vers une forme optimale représentant la complétude de l'ensemble des solutions d'une QBF.

Conjecture 1 *La combinaison de la recherche locale stochastique, de la généralisation de règle et de la réduction universelle permet de couvrir la complétude des solutions d'une QBF.*

Argument 1 *Soit une QBF Q et F sa formule propositionnelle sous-jacente, S_1, S_2 des scénarios de F tel que : S_1 falsifie F, S_2 satisfait F et S_1, S_2 n'appartiennent à aucunes solution de F. S_1 falsifie F, donc S_1 falsifie directement ou indirectement F. Si S_1 falsifie directement F en générant une clause vide, S_1 peut être généré par recherche locale stochastique et en appliquant ses substitutions sur F. Si S_1 falsifie indirectement F il est alors la généralisation de plusieurs règles d'élagage, il peut donc être détecté par la généralisation de règles.*
S_2 satisfait F, il ne peut donc être détecté par recherche locale stochatique et application de ses substitution qui satisferait alors F. S_2 n'est pas détectable pas la généralisation de règle construitent uniquement à partir de scénario falsifiant F. S_2 fait partie des non-solutions de F, il y a donc une universelle v instanciée dans S_2 pour laquelle il n'existe pas de scénario équivalent à S_2 jusqu'à v satisfaisant F pour l'autre valeur de v. Par généralisation on finira par couvrir l'ensemble des scénarios falsifiant F et par réduction universelle sur v on couvrira S_2.

3.7 Propagation des clauses unitaires

Toutes les méthodes énoncées précédemment peuvent être appliquées à une QBF qu'elle que soit sa forme, cependant le programme implémentant l'algorithme qui vous est présenté ici ne fonctionne actuellement que sur les QBF sous forme normale conjonctive (FNC), il est donc nécessaire d'introduire certains outils spécifiques à la résolution de ces formules.
Lorsqu'une clause ne contient qu'un littéral, elle est dite unitaire. En s'appuyant sur les travaux de [4], si la variable d'une clause unitaire est universellement quantifiée la formule est trivialement insatisfiable, si elle est quantifiée existentiellement on peut fixer sa valeur afin qu'elle valide la clause et propager cette substitution dans toute la formule.

4 L'algorithme

Le but de l'algorithme 1 est tout d'abord de générer le plus grand verrouillage possible d'existentielles, c'est à dire la combinaison de valeurs d'existentielles communes à toutes les solutions prouvant la validité de la QBF. En utilisant la propagation des clauses unitaires on obtient un premier verrouillage des variables, l'apprentissage de règles peut mener dans certains cas à des règles d'arité 1 correspondant également à un verrouillage.
La base de règle construite tend vers une forme optimale couvrant toute les non-solutions de la QBF, on tend donc vers la définition de la complétude des solutions de la QBF.
Une fois la base de règles d'élagages construite, ces règles sont utilisées lors de la résolution, lorsqu'une variable est instanciée, elles permettent de décider à l'avance de la valeur d'une partie des existentielles restantes. Cela permet d'éviter de tester des scénarios inutiles et de gagner du temps dans la résolution.

Algorithm 1 Apprentissage de la falsification

```
1: INPUT : <F,N> F une QBF et N un entier
2: OUTPUT : B une base de règles d'élagages

3: P une population de N scénarios
4: B une base de règles d'élagages
5: P ← N scénarios aléatoires falsifiant F
6: B ← ∅

7: B ← conversions des clauses en règles

8: repeat
9:    // Conversion de la population en règles
10:   B = apprentissage(P,B)
11:   // Comble les pertes de l'adaptation
12:   P = completer_population(P,N)
13:   // Génération génétique d'individus
14:   P = combine(P,F,N)
15:   // Comble les pertes de combine(P,F)
16:   P = completer_population(P,N)
17: until cas d'arrêt

18: return B
```

4.1 Commentaire de l'algorithme 1

Les premières règles construites par l'algorithme sont basées sur les clauses de la formule, les clauses étants des disjonctions il suffit que les variables de ses litteraux donnent faux pour falsifier la formule, on construit donc trivialement des règles en prenant l'inverse des litteraux des clauses.

Exemple 8 *Soit une formule F, R_1, R_2 des règles d'élagage tel que :*
$F = \forall a \exists b \forall c (a \vee b \vee \neg c) \wedge (\neg b \vee c)$
$R_1 = [a = 0,\ b = 0,\ c = 1]$ déduite de $(a \vee b \vee \neg c)$
$R_2 = [b = 1,\ c = 0]$ déduite de $(\neg b \vee c)$

4.2 Adaptation

L'adaptation consiste en la généralisation des scénarios couverts par la base de règles, les variables à généraliser sont selectionnées en fonction des règles qui couvrent un scénario (ligne 5). Pour généraliser une variable on change sa valeur pour son inverse et si le scénario falsifiie toujours la formule alors le scénario est généralisable en cette variable. En adaptant les scénarios aux règles on contribue à la création de nouvelles règles et à la révision des règles existantes. Cela permet d'écarter les scénarios qui ne falsifient F que parce qu'ils sont l'expression de règles déjà existantes (ligne 11). La création des règles d'élagages est le fruit d'une recherche locale stochastique guidée par les règles produites par les générations de scénarios précédentes.

Algorithm 2 Adaptation

```
1:  INPUT : <S,B> S un scénario, B une base de règles
2:  OUTPUT : S un scénario

3:  for chaque règle R qui couvre S do
4:     for chaque littéral v ← val de R do
5:        if [v ← ¬val](S) falsifie F then
6:           S = [v ←?](S)
7:           break //passer à la prochaine règle
8:        end if
9:     end for
10:    return false
11: end for
12: return true

13: return P
```

Exemple 9 *Soit S un scénario falsifiant une formule F, R_1 une règle couvrant S tel que :*
$F = \forall a \exists b \forall c \exists d (b \vee d) \wedge (a \vee b)(\neg b \vee c)$
$S = [0,0,?,0]$
$R = [b = 0,\ d = 0]$ couvre S car la valeur de b et d dans S est 0 comme spécifié dans la règle.
On tente d'adapter S en switchant les valeurs des variables concernées par R : $S_1 = [b \leftarrow 1](S) = [0,1,?,0]$, en appliquant S_1 sur F il reste la clause unitaire (c), si $[c \leftarrow 1]$ F vaut vrai, S_1 ne falsifie donc pas F et S n'est pas généralisable en b.
On tente alors de changer la valeur de d, $S_2 = [d \leftarrow 1](S) = [0,0,?,1]$ falsifie $(b \vee d)$ donc falsifie F, S est donc généralisable en d et S ne sera plus couvert par R.

4.3 Apprentissage

Algorithm 3 Apprentissage

```
1:  INPUT : <P,B> P une population de scénarios, B une base de règles
2:  OUTPUT : B une base de règles
3:  G une liste de règles
4:  G ← ∅

5:  for chaque scénario S de P do
6:     // Adaptation du scénario aux règles
7:     if adaptation(S,B,F) then
8:        R ← conversion de S en règle

9:        // suppression des règles couvertes par R
10:       for chaque règle R' de B d'arité ≥ R do
11:          if R couvre R' then
12:             retirer R' de B
13:          end if
14:       end for

15:       // Généralisation des règles de même arité
16:       for chaque règle R' de B de même arité que R do
17:          if R et R' ne diffèrent que sur la valeur d'un littéral then
18:             ajouter la généralisation de R et R' à G
19:             retirer R' de B
20:          end if
21:       end for

22:       ajouter de la même façon les généralisations stockées dans G
23:    end if
24: end for

25: return B
```

Une fois un scénario adapté à la base, il se traduit par une nouvelle règle. Lors de l'ajout de cette règle on supprime toutes les règles qu'elle subsume (ligne 7 à 12).
La base est ensuite utilisée pour trouver des généralisations de la règle (ligne 14 à 21), si il y en a, la règle est supprimée et les généralisations ajoutées de la même façon à la base, sinon la règle est placée dans la base.

Dans le cas d'une règle d'arité 1, nous sommes en présence d'un verrouillage d'existentielle, les litteraux portant sur cette variable sont alors retirés de toutes les règles de la base et la formule simplifiée.
Si une règle universelle (règle ne contenant que des litteraux portant sur des variables universellement quantifiées) est ajoutée à la base cela signifie que la formule est insatisfiable puisque falsifiée par cette instanciation d'universelles, la QBF est donc non valide. L'algorithme s'arrête et retourne cette règle comme preuve de la non validité de la QBF.

4.4 Génération des scénarios

L'algorithme génère des scénarios aléatoires falsifiant la formule, de ceux-ci sont déduits des règles d'élagages. En suivant les principes d'un algorithme génétique les scénarios ayant générés des règles sont conservés pour créer une nouvelle génération de scénarios.

Algorithm 4 Combine

```
1: INPUT : <P,F,N> P une population de scénario, F une QBF, N un entier
2: OUTPUT : P une population de scénarios

3: while N combinaisons n'ont pas été effectuées do
4:    male ← un individu choisit au hasard parmi la moitié supérieure de P
5:    female ← un individu choisit au hasard dans P
6:    rupture ← un entier choisit au hasard entre 0 et le nombre de variables de F
7:    descendant ← un scénario vide
8:    ajouter à descendant les éléments de male de 0 à rupture
9:    ajouter à descendant les éléments de female de rupture à nombre de variables dans F
10:   if descendant falsifie F then
11:      ajouter descendant à P
12:   end if
13: end while

14: return B
```

Exemple 10 *Soit S_1, S_2, S_3 des scénarios falsifiant une même formule F tel que :*
S_1 = [1,1, ?,0]
S_2 = [0,1,1, ?]
S_3 = [1,1,1, ?] est la combinaison génétique de la moitié inférieure de S_1 et de la moitié supérieure de S_2. Si S_3 falsifie la formule propositionnelle de F il sera conservé comme individu de la nouvelle génération. S_3 est une nouvelle combinaison de variables, en combinant S_1 et S_2 on espère transmettre les gènes, comprenez les valeurs de variables qui leur ont permis de survivre et de devenir des règles inédites à la base. Dans notre programme le point de rupture du génome des parents est choisit aléatoirement lors de chaque combinaison d'individus.

L'algorithme utilise deux méthodes pour générer des scénarios, la première est une instanciation aléatoire des variables du scénario. La génération aléatoire d'individu peut lever un cas d'arrêt, si aucun scénario n'a pu être créé après un nombre de tentative fixé on peut estimer que la formule n'est plus falsifiable.
La seconde utilise les principes des algorithmes génétiques pour créer de nouveaux individus à partir d'une population de scénarios ayant permis la création de règles d'élagage. La combinaison génétique permet de créer des scénarios d'une façon plus rationnelle que la simple génération aléatoire, les combinaisons de variables falsifiant la formule sont transmises et mélangées dans les descendants ce qui permet l'apparition de nouvelles règles ou l'améliorations de règles existantes.

5 Résolution

Algorithm 5 Resolution

```
1: INPUT : <F,B> F une QBF, B une base de règles
2: OUTPUT : valide / non valide

3: V la première variable de F
4: S un scénario initialisé avec les variables fixée de F

5: if V != ? dans Scenario then
6:    return resoudre(Scenario,F,B)
7: end if

8: val la valeur la moins présente dans les scénarios couvert par B

9: if V est quantifiée existentiellement then
10:   [V ← val](S)
11:   if resoudre(0,S,F,B) then
12:      return valide
13:   else
14:      [V ← 1](S)
15:      return resoudre(0,S,F,B)
16:   end if
17: else
18:   // V quantifiée universellement
19:   [V ← ¬val](S)
20:   if resoudre(N,S,F,B) then
21:      [V ← 1](S)
22:      return resoudre(0,S,F,B)
23:   else
24:      return non valide
25:   end if
26: end if
```

Une fois la base de règle construite et jugée suffisante (en fonction du critère d'arrêt) elle peut être utilisée

pour guider la recherche de solutions lors de la résolution de la formule.

Pour prouver la validité de la formule on doit trouver une combinaison d'exitentielle satisfiant la formule pour chaque combinaison possible d'universelles. On instancie donc les variables dans l'ordre des quantificateurs tant qu'on ne falsifie pas la formule, lorsque le scénario courant satisfait la QBF on remonte une solution potentielle jusqu'a la dernière universelle rencontrée, on cherche ensuite une solution pour son autre valeur, si toutes les valeurs d'une universelle mènent à une solution potentielle on revient en arrière jusqu'a l'universelle précédente et ainsi de suite. Si au contraire le scénario falsifie la formule on remonte jusqu'à la dernière existentielle non fixée et pour laquelle il reste une valeur non testée.

5.1 Déduction de valeur

Les règles d'élagage permettent de détecter la falsification de la formule à l'avance, si le scénario courant est couvert par la règle il est inutile de continuer avec cette instanciation de variable, le retour-arrière s'impose. Les règles permettent de fixer à l'avance la valeur des variables, si le scénario courant est couvert par une règle privée d'un élément, on fixe alors la valeur de la variable de cet élément dans le scenario pour qu'il ne soit pas couvert par la règle dans la suite de la résolution.

Exemple 11 *Soit R une règle d'élagage et S le scénario représentant la solution potentielle courante :*
S = [0,0,0, ?, ?]
R = [a = 0, b = 0, e = 0]
a, b et c sont à 0 dans S, S est couvert par R privé de son dernier littéral (e = 0), donc si on met e à 0 dans S on falsifiera la formule propositionnelle de F selon la définition des règles d'élagage. On peut donc fixer la valeur de e à 1 dans S.
S_1 *= [e←1](S) = [0,0,0, ?,1]*

5.2 Heuristique d'élagage

Le nombre de scénarios couvert par la base de règles peut être utilisé comme une heuristique lors du choix de la valeur de la variable courante. On connait le nombre de scenarios couverts par la base pour une valeur de variable donnée, il est donc statistiquement plus probable de falsifier la formule si la variable prend la valeur la plus présente dans les non-solutions.

Lorsque la variable courante est une existentielle, on choisira donc en premier la valeur la moins présente dans les non-solutions afin de tendre plus rapidement vers une solution, pour une universelle on choisira en priorité la valeur la plus présente dans les non-solutions afin de falsifier la formule rapidement et d'effectuer un retour-arrière plus tôt.

Algorithm 6 Resoudre

```
1: INPUT : <N,S,F,B> N un entier, S le scenario courant, F une QBF, B une base de règles
2: OUTPUT : Satisfiable un booléen

3: // Application des règles
4: for chaque règle R de B do
5:    if R privé de son dernier littéral [L = val] couvre S then
6:       // Incohérence
7:       if la variable L = ¬val dans S then
8:          return non valide
9:       else
10:         [L ← val](S)
11:      end if
12:   end if
13: end for

14: // Test du scénario
15: if S statisfait F then
16:    return valide
17: end if
18: if S falsifie F then
19:    return non valide
20: end if

21: if V != ? dans Scenario then
22:    return resoudre(Scenario,F,B)
23: end if

24: val la valeur la moins présente dans les scénarios couvert par B

25: // Récursivité
26: V la N+1-ième variable de F dans l'ordre des quantificateurs

27: if V est quantifiée existentiellement then
28:    [V ← val](S)
29:    if resoudre(N+1,S,F,B) then
30:       return valide
31:    else
32:       [V ← 1](S)
33:       return resoudre(N+1,S,F,B)
34:    end if
35: else
36:    // V quantifiée universellement
37:    [V ← ¬val](S)
38:    if resoudre(N+1,S,F,B) then
39:       [V ← 1](S)
40:       return resoudre(N+1,S,F,B)
41:    else
42:       return non valide
43:    end if
44: end if
```

5.3 Incohérences

Dans un premier temps les règles permettent de déduire les valeurs des prochaines existentielles, cependant au cours de la résolution il est possible que deux règles qui se terminent sur une valeur différente de la même existentielle soient applicables. Il s'agit alors d'une incohérence, cela signifie que peu importe la valeur de la variable dans le scénario actuel, la formule propositionnelle sera falsifiée par le scénario.

Exemple 12 *Soit R_1 et R_2 deux règles d'élagage tel que :*
$R_1 = [a = 0,\ c = 0,\ d = 0]$
$R_2 = [b = 0,\ d = 1]$

Lorsque dans la résolution on rencontre un cas où $a = 0$, $b = 0$ et $c = 0$ on détecte une incohérence sur d, en mettant b à 0 on déduit de R_2 $d = 0$, puis en mettant c à 0 on déduit de R_1 $d = 1$, peut importe la valeur de d le scénario falsifiera la formule, on effectue un retour arrière sans avoir à évaluer la formule.

6 Résultats experimentaux

L'utilisation des principes des algorithmes génétique est ici assez particulière, pour en expliquer l'intérêt il convient de rappeler le cheminement qui à conduit à sa forme actuelle.

6.1 Recherche locale stochastique

Au départ nous utilisions la recherche locale stochastique pour faire muter les individus, par muter nous entendons généraliser les scénarios. Suite à plusieurs mutations aléatoires les scénarios étaient convertis en règle, puis la règle ajoutée à la base si elle n'était pas subsumée.
La recherche locale stochastique seule n'apportait pas de résultats concluant, la combinaison génétique des individus ayant créer des règles apporta son lot d'améliorations. L'évolution de la base de règles n'était pas encore satisfaisante, l'adaptation s'imposa donc permettant aux scénarios de muter selon les règles et d'enrichir un peu plus la base de règles d'élagage.
Se posa ensuite la question de l'éfficacité, une mutation pour être acceptée, requière de tester entièrement le scénario sur la QBF, de plus sans les mutations, l'adaptation permettait de créer les mêmes règles en quelques générations suplémentaires. Les mutations stochastiques pures, furent donc abandonnées au profit de celles guidées par l'adaptation.

6.2 Approche génétique

Dans notre approche génétique les individus doivent survivre en s'adaptant aux règles créées par leurs ancêtres, ils n'ont donc que deux façons de survivre et

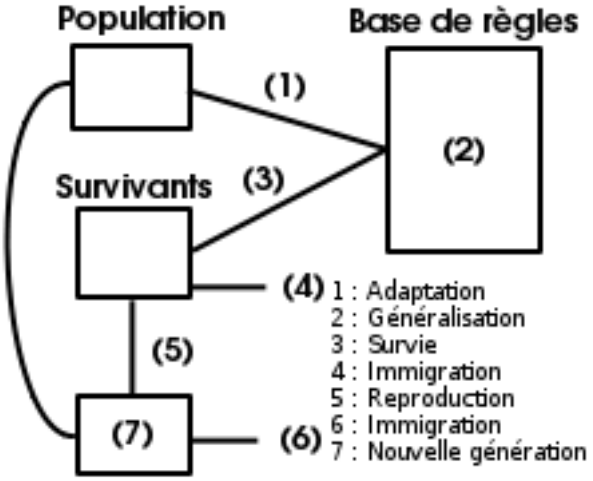

FIGURE 2 – Représentation schématique de notre algorithme génétique

de pouvoir transmettre leur patrimoine génétique. Premièrement ils peuvent innover, s'ils ne sont couverts par aucune règle ils permettent l'apprentissage d'une règle inédite à la base, s'ils sont couverts par des règles, ils peuvent muter pour aboutir ici aussi à une règle inédite. Sinon ils peuvent améliorer les règles de leurs ancêtres, si un scénario est l'expression même d'une règle connue et qu'ils s'y adaptent, il produira alors une règle subsumant cette dernière, ce scénario améliore donc la base en généralisant les règles existantes. La sélection "naturelle" est de plus en plus dure, chaque génération réduisant les chances de survie de ses descendants, qui doivent en conséquence évoluer pour passer le relais.

6.3 Apprentissage interne

Les règles n'avaient toujours pas donner tout leur potentiel, en comparant des règles de même arité nous avons constater que certaines ne différaient que d'un litteral et pouvaient être généralisées tout comme les scénarios (cf. section 3.3 page 3). La généralisation de règles permis la création de règles qui ne pouvaient être déduites des scénarios, en effet dans l'implémentation que nous avons fait de notre algorithme, le test de falsification d'une formule par un scénario ne teste que les valeurs instanciées du scénario. Vérifier pour chaque scénario toutes les combinaisons de valeur de ses variables génériques (?) ne serait pas raisonnable en temps de calcul. Grace à la généralisation de règles la base de règles tend vers une forme optimale, couvrant toutes les non-solutions et composée des règles les plus petites possibles en nombre de litteraux (les plus générales). Or il arrive que des règles bloquent la création d'autres règles qui permettraient des généralisations dans la base, lorsqu'un scénario est le plus

générique possible et qu'il est couvert par une règle il ne peut être adapté à cette règle avec la méthode de test de falsification expliquée précédemment.

Exemple 13 *Soit F une QBF, R_1, R_2, R_3, R_4, R_5 des règles d'élagage portant sur F tel que :*
$F = \forall a \exists b \exists c (a \vee b) \wedge (a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (a \vee \neg b \vee \neg c) \wedge (a \vee b \vee \neg c)$
$R_1 = [a = 0, b = 0]$ *déduite de* $(a \vee b)$
$[a = 0, b = 0, c = 0]$ *déduite de* $(a \vee b \vee c)$ *est subsumée par* R_1.
$R_2 = [a = 0, b = 1, c = 0]$ *déduite de* $(a \vee \neg b \vee c)$.
$R_3 = [a = 0, c = 0]$ *généralisation de* R_1 *et* R_2 *en b.*
$R_4 = [a = 0, b = 1, c = 1]$ *déduite de* $(a \vee \neg b \vee \neg c)$.
$R_5 = [a = 0, b = 0, c = 1]$ *déduite de* $(a \vee b \vee \neg c)$ *est subsumée par* R_1*, on obtiendra donc jamais* $[a = 0, b = 1]$ *par généralisation de* R_4 *et* R_5 *qui se généraliserait avec* R_1 *pour donner* $[a = 0]$.
$S = [a = 0, b = 1]$ *sans propagation des clauses unitaires ne falsifie aucune clause de F et ne sera donc pas conservée.*
Ici R_1 *bloque sa propre généralisation, mais une règle peut très bien bloquer la généralisation d'autres règles.*

Comme l'illustre cet exemple il est important de rechercher des généralisations avant l'adaptation pour éviter que des règles ne bloquent des généralisations.

6.4 Résultats

Les QBF utilisées dans cette section proviennent de QBFLIB (qbflib.org). La résolution est effectuée plusieurs fois sur les petites QBF pour refléter le gain ou la perte de temps dû à l'utilisation des règles d'élagage. L'apprentissage des règles est effectué avant la résolution en un temps fixé, ce temps est indiqué dans la colonne "temps d'apprentissage". Dans l'ordre des colonnes nous avons, le nom de la QBF (QBF), le nombre de variables (Var), le nombre de clauses (Cla), le nombre d'individus d'une génération d'apprentissage (Pop), le temps d'apprentissage (App), le nombre de règles construites (Règles), le nombre d'existentielles fixées (Fixes), le nombre de déduction (cf. section 5.1 page 8) effectuer lors de la résolution (Déd), le nombre d'incohérences détectées (cf. section 5.3 page 9) (Inc), le nombre de résolutions effectuées (Nb Réso), le temps de résolution avec base de règles (Réso A) et le temps de résolution sans apprentissage (Réso S).
Les tests ont tous été effectués sur la même machine, processeur Intel(R) Core(TM)2 Duo CPU P8400 2.26GH, 2992 Mo de ram et nous avons utilisé le logiciel sKizzo [2] pour vérifier la cohérence de nos résultats.
Pour **AB** et **toilet_g_02_01.2-shuffled** l'algorithme construit une base estimée comme optimale en utilisant uniquement les règles déduites des clauses de la formule. La base est jugée optimale quand il devient impossible de créer un scénario falsifiant la formule et non couvert par la base, dans le programme ce cas est représenté par un seuil maximum d'échecs consécutifs lors de la création de scénarios, si le seuil est atteint le programme s'arrête.
Pour **impl02** l'algorithme fixe toutes les existentielles 9 fois sur 10 se qui correspond au temps 0 de résolution, dans les autres cas le temps de résolution est inférieur à 25% du temps de résolution sans base de règles.
Sur **impl04** on observe une amélioration du temps de résolution de 35% à 50%, en dessous de 1 seconde d'apprentissage la base de règles ne permet une amélioration de 40% à 60% qu'une fois sur deux. Avec 1 seconde d'apprentissage, le temps de résolution tourne autour de 0.25 secondes et l'algorithme fixe toutes les existentielles à true une fois sur dix ce qui correspond à un temps de résolution de 0 secondes.
En ce qui concerne **tree-exa10-10**, les tests ont été jusqu'à 30 secondes d'apprentissage, la base de règle n'a de cesse de grandir au fil du temps et le nombre de déductions et d'incohérences détectées reste le même, à savoir 18 déductions et aucunes incohérences. Dans cette QBF toutes les universelles précèdent les existentielles, de ce fait la réduction universelle n'est jamais appliquée, la base de règle n'apporte quasiment rien dans la résolution de cette QBF.
Le même comportement de la base de règles s'observe sur **toilet_g_15_01.2**, la quantité de règle va en augmentant mais le nombre de déductions et d'incohérences reste le même. Le coût d'utilisation de la base de règles est ici supérieur au temps gagné par ses déductions et ses détections d'incohérences.
Le cas **toilet_c_04_10.2** est interressant car il montre l'intérêt de la détection d'incohérence, permettant un retour-arrière prématuré. Le temps de résolution est d'environ 1% du temps initial, sur cette exemple la partie génétique n'est pas exploitée puisqu'au delà de 10 individus par génération on a 1 seconde par génération. Lors des tests nous avons obtenue une base de 703 règles avec 1 seconde d'apprentissage, elle a permis à la résolution de ne durée que 0.04 secondes, ce résultat n'as été obtenue qu'une fois sur une trentaine de tests sur cette QBF.
La QBF **robots_1_5_2_99.1** montre les limites du programme, le nombre de variable et de clauses fait qu'une génération de 4 scénarios met plus de 2 secondes pour être traitée, les 1976 clauses de la formule ont construit la grande majorité des règles contenue dans la base en sortie. En attribuant plus de temps d'apprentissage la base de règles s'agrandit sans donner de meilleurs résultats lors de la résolution, en augmentant le temps d'apprentissage et le nombre d'individus d'une génération la probabilité d'obtenir une base telle que celle-ci augmente, ces bases doivent couvrir une grande partie de la complétude des solutions de la QBF.

En effectuant juste un apprentissage sur les clauses de la QBF **robots_1_5_2_99.1** qui dure 0.54 secondes, la résolution basée sur la conversion de la base de règles en fichier qdimacs par sKizzo passe de 0.12 à 0.04, certe le temps d'apprentissage compense le gain de temps, mais cela montre néanmoins que notre prétraitement peut améliorer la résolution de QBF par d'autre solveurs.
En générale dans le pire des cas, l'utilisation de la base de règles augmente de 10% le temps de résolution, les améliorations ne dépendent pas du nombre de variables ou de clauses comme l'illustre les résultats obtenuent sur **toilet_g_15_01.2** et **toilet_c_04_10.2**. La forme de la QBF est un facteur important, tant sur le temps d'apprentissage que sur le temps de résolution et d'utilisation de la base de règles. La base de règle tend dans certains cas vers une forme optimale couvrant la totalité des non-solution, c'est le cas pour **AB**, **toilet_g_02_01.2** et sans doute également pour **toilet_c_04_10.2**, dans ce cas le temps de résolution est grandement amélioré.

QBF	Var	Cla	Pop	App	Règles	Fixes	Déd	Inc	Nb Réso	Réso A	Réso S
AB	5	9	20	0.02	9	0	2	0	100 000	2.16	3.05
					9	0	2	0		2.17	
					9	0	2	0		2.14	
					9	0	2	0		2.15	
					9	0	2	0		2.18	
impl02	10	18	20	0.03	8	8	0	0	100 000	**0**	13.10
				0.34	46	1	7	0		1.58	
				0.04	8	8	0	0		**0**	
				0.45	74	1	5	0		3.25	
				0.28	50	1	11	0		2.48	
toilet_g_02_ 01.2-shuffled	11	22	20	0.01	14	8	2	0	100 000	0.7	1.8
					14	8	2	0		0.69	
					14	8	2	0		0.71	
					14	8	2	0		0.72	
					14	8	2	0		0.7	
impl04	18	34	20	1	830	1	12	0	1 000	**1.18**	2.26
					830	1	12	0		1.46	
					1 181	1	40	0		1.44	
					1 043	1	47	0		1.76	
					929	1	28	0		1.32	
tree-exa10-10	20	18	20	0.05	89	0	18	0	1 000	3.67	3.65
					126	0	18	0		3.70	
					145	0	18	0		3.66	
					106	0	18	0		3.50	
					103	0	18	0		3.45	
toilet_g_15_ 01.2	79	360	4	1.03	190	30	342 672	16 352	1	29.95	27.12
				1.53	244	30	342 672	16 352		30.87	
				3.27	300	30	342 672	16 352		32.54	
			10	10.8	1602	30	340 137	16 382		41.88	
			10	29.13	2846	30	342 672	16 352		62.42	
toilet_c_04_ 10.2	138	850	4	0.51	676	28	1 850	332	1	**0.66**	7.38
				0.5	683	28	1 850	332		**0.65**	
				0.56	673	28	1 850	332		**0.67**	
				0.52	670	28	1 850	332		**0.63**	
				0.5	672	28	1 850	332		**0.65**	
robots_1_5_ 2_99.1	1976	5006	10	**3.17**	1099	37	119 051	0	1	32.55	29.05
				2.99	1 097					32.43	
				1.93	1 098					32.76	
				2.12	1 098					32.25	
			10	**30**	1 222					59.26	

7 Conclusion

Nous avons exposé dans cette article, l'utilisation qui peut être faite de l'insatisfiabilité de la formule propositionnelle sous-jacente d'une formule booléenne quantifiée, pour déterminer sa validité. Exploiter l'insatisfiabilité, permet de réduire l'espace de recherche lors de la recherche de la validité de la QBF correspondante. En exploitant les différentes propriétés propres à la falsification nous sommes aussi en mesure d'améliorer la recherche de solutions pour des QBF de forme quelconque. Au vue des tests effectués, l'algorithme permet d'améliorer notablement la résolution de certaines QBF, dans le pire des cas la résolution est plus longue, ne dépassant généralement pas 110% du temps de calcul sans la méthode.

Notre méthode d'apprentissage peut permettre de fixer la valeur de certaines variables dans la QBF. Pour des QBF de forme générale elle pourrait tenir la place de la propagation des clauses unitaires qui se limite à la forme normale conjonctive. Il reste de nombreux points à améliorer : des optimisations d'ordre structurelle pour le programme ou l'utilisation de méthodes telle que l'exploitation des litteraux monotones.

La méthode d'apprentissage par généralisation qui vous est présentée peut s'appliquer à d'autres domaines finis que celui des booléens, une fois toutes les valeurs testées pour une variable on peut en déduire sa généralisation dans le scénario concerné et apprendre des règles d'inférence.

7.1 Conclusion Personnel

La première raison qui m'a poussé à choisir ce sujet, c'est qu'il m'offrait un premier aperçus d'un travail de recherche, je nourrissais l'espoir d'en apprendre beaucoup, d'une part sur le domaine des QBF et sur les différentes étapes qui constituent un travail de recherche, je n'ai pas été deçu. J'ai approfondi mes connaissances en logique, compris des aspects que j'étais loin de maitriser, saisi l'importance du formalisme lorsqu'on veut expliquer ses travaux. J'ai également appris les bases de la rédaction d'un article de recherche, fais connaissance avec certains outils utilisés par la communauté tel que Latex, avec lequel j'ai rédigé ce rapport.

Ce que j'ai préféré c'est la liberté, une grande partie du travail que j'ai réalisé est le fruit de mon imagination. Mon maitre de stage m'a aidé à formaliser, identifier les problèmes que j'ai rencontré au fil du stage, cela nous a permis de trouver des solutions adéquates, de constater sur quoi aboutissait réellement les méthodes utilisées. Les articles de recherche et autres lectures qu'il m'a fournit m'on été d'un grand secours, tant pour résoudre des problèmes que pour trouver des améliorations.

Ce que je pense avoir le mieux réussi c'est l'exploitation de la falsification, j'ai mis au point un algorithme utilisant des méthodes simples, permettant des généralisations en chaine et l'apprentissage de ce qui caractérise la complétude des solutions d'une formule booléenne quantifiée. Pour faire un parallèle avec l'apprentissage artificiel notre méthode permet de définir par des règles d'inférence tout ce qui n'est pas une solution. Il est loin d'être parfait, mais il a au moins le mérite de montrer que la falsification d'une formule propositionnelle peut être exploitée pour aider à résoudre une QBF, ce qui était l'objectif de base du stage.

Ce qui m'a demandé le plus d'effort c'est la rédaction de l'article, réussir à expliquer les méthodes utilisées, démontrer leurs propriétés et ce en respectant un formalisme compréhensible et surtout correct n'est pas une mince affaire. Je n'ai pas l'habitude d'être aussi formel, chaque terme a son importance, chaque symbole doit être défini. Ce qui m'apparaissait aux premiers abords être une corvée, s'est avéré une excellente source d'inspiration, en illustrant mes propos par des exemples, j'ai pu identifier certains cas particuliers et pu mettre au point des parades. Il y a donc une certaine rigueur qui me fait toujours défaut, ainsi qu'un manque de connaissance du domaine qui m'as un peu handicapé au début du stage.

Ce stage m'as permis d'accroitre mes connaissances sur les propriétés des QBF. J'ai pu découvrir de nouvelles méthodes tel que les algorithmes génétiques ou la recherche locale stochastique et surtout appris à exploiter leur potentiel. Cette expérience conforte mon choix de poursuivre mes études dans la recherche, j'ai pris beaucoup de plaisir à acquerir de nouvelles connaissances et à me triturer l'esprit pour trouver des méthodes permettant de résoudre les différents problèmes rencontrés. Je tiens à remercier mon maitre de stage monsieur Igor Stéphan d'avoir eu la gentillesse et la patience de m'expliquer nombre de points techniques propre aux domaines qu'il m'était difficile de comprendre et d'avoir suivi mes travaux tout au long du stage, me permettant chaque semaine de vérifier si je ne faisais pas fausse route.

7.2 Bibliographie

J'ai appris beaucoup des articles fournit par mon maitre de stage, vous les retrouverez dans la partie bibliographie en fin d'article. Wikipédia pour ce qui est des principes des algorithmes génétiques (http ://fr.wikipedia.org/wiki/Algorithme_génétique), le site commentcamarche.net pour certaines spécifités liées à l'utilisation de Latex (http ://www.commentcamarche.net/contents/latex/latex-caracteres.php3). Pour la programmation j'ai eu recours à la documentation de cplusplus.com (http ://www.cplusplus.com), pour ce qui est de la syntaxe des fichiers qdimacs je me suis basé sur la description qui en faite sur qbflib.org (http ://www.qbflib.org/).

Références

[1] Vincent barichard and Igor Stéphan. La contrainte d'équivalence dans qcsp(qbf). In *Actes JFPC*, 2011.

[2] M. Benedetti. sKizzo : a QBF Decision Procedure based on Propositional Skolemization and Symbolic Reasoning. Technical Report 04-11-03, ITC-irst, 2004.

[3] M. Benedetti. Abstract branching for quantified formulas. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI'06)*, 2006.

[4] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An Algorithm to Evaluate Quantified Boolean Formulae and Its Experimental Evaluation. *Journal of Automated Reasoning*, 28(2), 2002.

[5] Niklas Eén and Armin Biere. Effective preprocessing in sat through variable and clause elimination. In *8th International Conference on Theory and Applications of Satisfiability Testing*, 2005.

[6] I.P. Gent, H.H. Hoos, A.G.D. Rowley, and K. Smyth. Unsing Stochastic Local Search to Solve Quantified Boolean Formulae. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP'03)*, 2003.

[7] Cédric Piette, Youssef Hamadi, and Lakhar Saïs. Vivification de formules propositionnelles clausales. In *Quatrièmes Journées Francophones de Programmation par Contraintes*, Nantes, France, 2008.

[8] Pnot Pipatsrisawat and Adnan Darwiche. *RSAT 2.0 : SAT solver description. Technical Report D-153.* Automated Reasoning Group, Computer Science Department, 2007.

[9] Igor Stéphan and Benoit Da Mota. Base littérale et certificat pour les formules booléennes quantifiées. In *Quatrièmes Journées Francophones de Programmation par Contraintes*, Nantes, France, 2008.